CSCI 3753: Operating Systems

Week 4
September 19, 2025

University of Colorado **Boulder**

Announcements

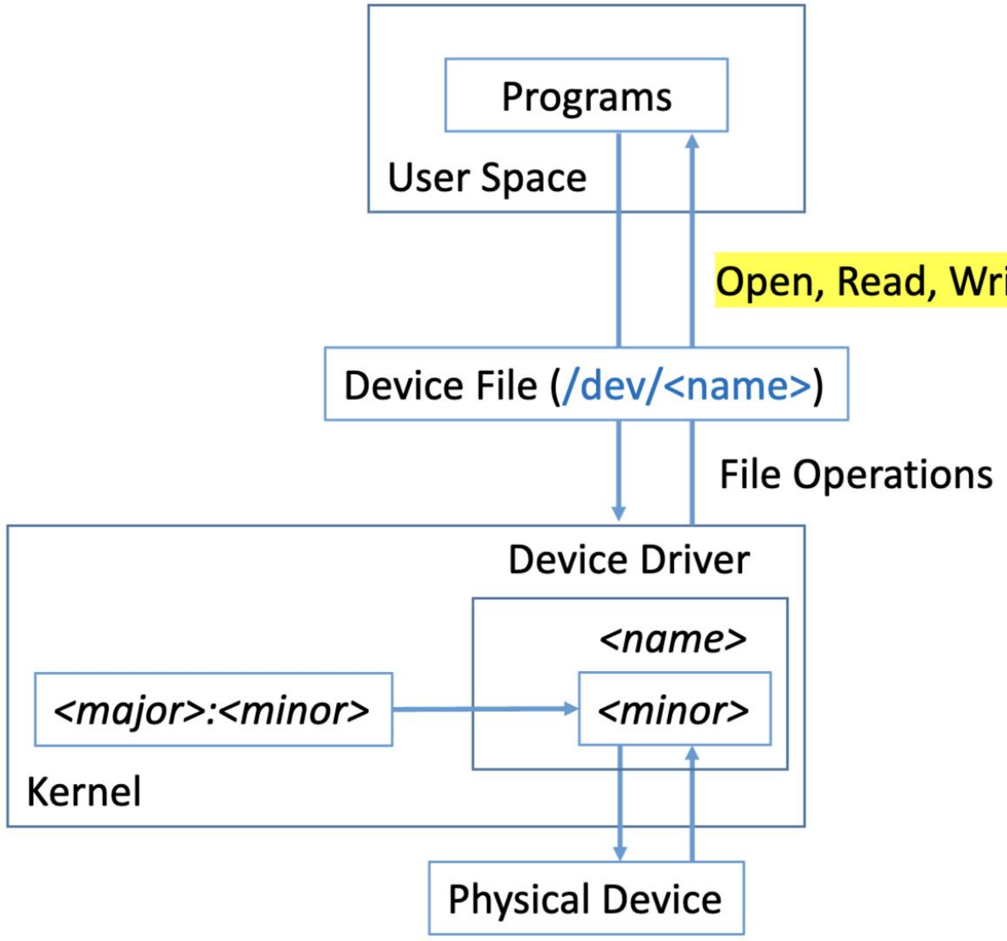
- PA2 due Sunday at midnight!
- Quiz 4 due at midnight

- Recitation materials: https://tinyurl.com/CSCI3753

Recap

- PA0+1 Interview Grading ✅
- Problem Set 1
- Introduced:
  - Processes
  - Threads
- PA2+3 🔁

PA 3


Programs
User Space
Open, Read, Write, Close, Seek
Device File (/dev/<name>)
File Operations
Device Driver
<name>
<major>:<minor>
<minor>
Kernel
Physical Device

```
mknod –m <permission>
         <device_file_location>
         <type of driver>
         <major number>
         <minor number>
```

PA 3

- create a Device Driver Module (LKM)
- implement file operations
  - open, seek, read, write, release
- make and load the module
- create a Device File for this Device

PA 3

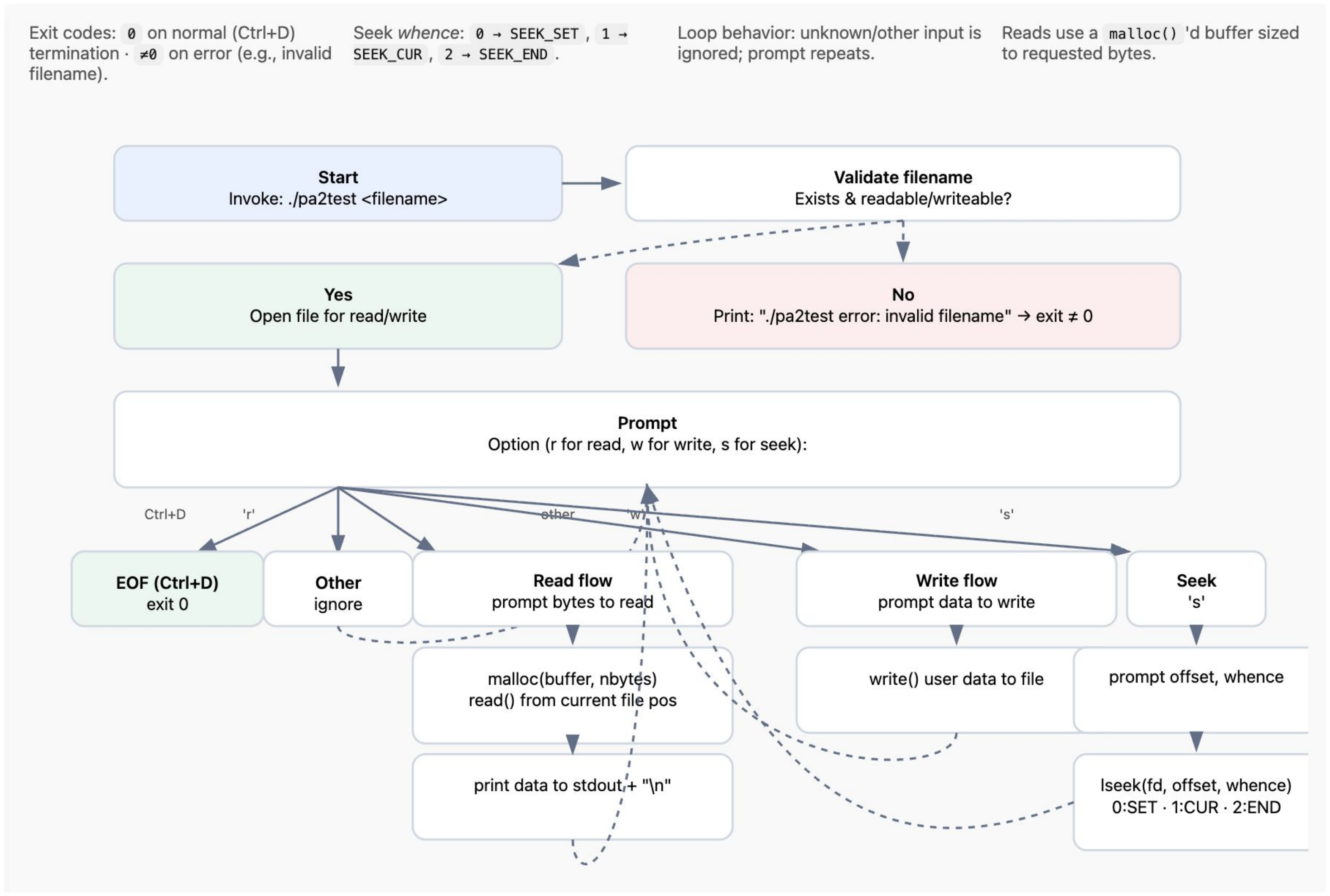
- create a Device Driver Module (LKM)
- implement file operations
  - open, seek, read, write, release
- make and load the module
- create a Device File for this Device
- **create a test program**

PA 2

PA 2

- Test program for PA 3
- pa2test.c
  - infinite loop with the following features
    - r - read()
    - w - write()
    - s - seek()
      - SEEK_SET
      - SEEK_CUR
      - SEEK_END
    - control+d for termination
    - other entries should be ignored

PA 2 Flow


Exit codes: `0` on normal (Ctrl+D) termination · `≠0` on error (e.g., invalid filename).

Seek *whence*: `0 → SEEK_SET`, `1 → SEEK_CUR`, `2 → SEEK_END`.

Loop behavior: unknown/other input is ignored; prompt repeats.

Reads use a `malloc()`'d buffer sized to requested bytes.

Start Invoke: ./pa2test <filename>
Validate filename Exists & readable/writeable?
Yes Open file for read/write
No Print: "./pa2test error: invalid filename" → exit ≠ 0
Prompt Option (r for read, w for write, s for seek):
Ctrl+D
'r'
other
'w'
's'
EOF (Ctrl+D) exit 0
Other ignore
Read flow prompt bytes to read
malloc(buffer, nbytes) read() from current file pos
print data to stdout + "\n"
Write flow prompt data to write
write() user data to file
Seek 's'
prompt offset, whence
lseek(fd, offset, whence) 0:SET · 1:CUR · 2:END

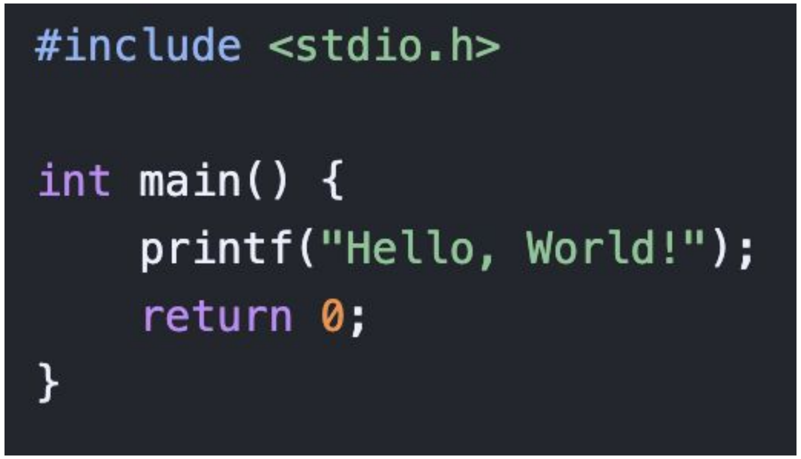
PA2 Demo

- Available on Canvas

C Input + Output

C Output

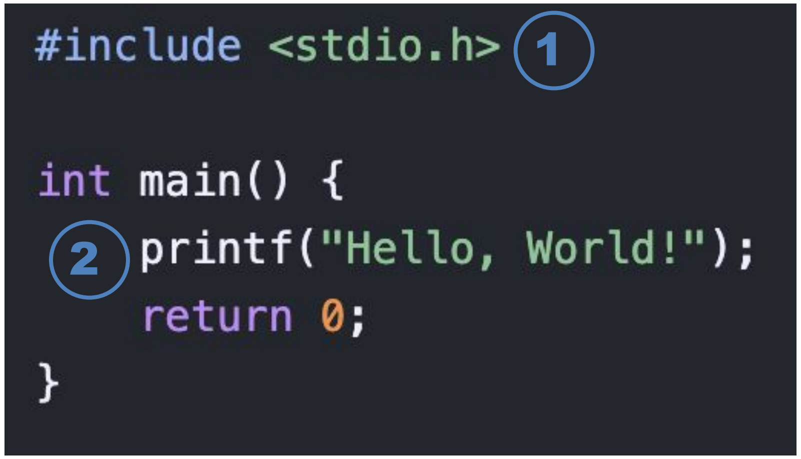
- Two pieces
  1. library
  2. print statement



```c
#include <stdio.h>

int main() {
    printf("Hello, World!");
    return 0;
}
```

C Output

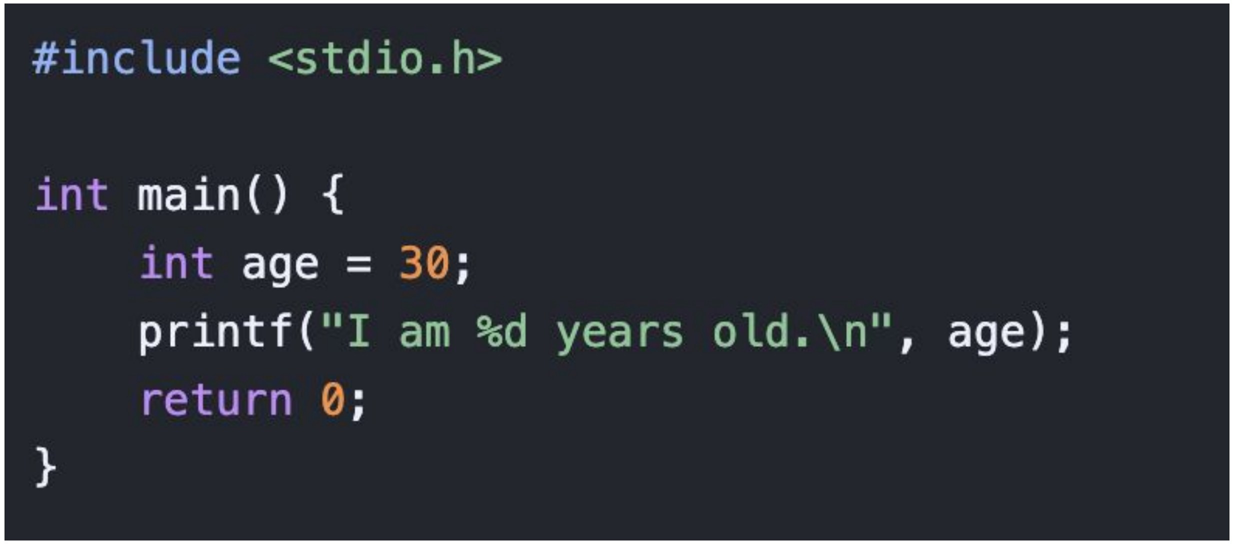
- Two pieces
  1. library
  2. print statement



```
#include <stdio.h>

int main() {
    printf("Hello, World!");
    return 0;
}
```

C Output

- You can also format output with printf()



```c
#include <stdio.h>

int main() {
    int age = 30;
    printf("I am %d years old.\n", age);
    return 0;
}
```

C File Input + Output

- For testing purposes it can be useful to:
  - Input a files worth of text into a program
    - ./pa2test < commands.txt
  - Output the text to a file:
    - ./pa2test > output.txt

C File Input + Output

- For testing purposes it can be useful to:
    - Input a files worth of text into a program
        - ./pa2test < commands.txt
    - Output the text to a file:
        - ./pa2test > output.txt

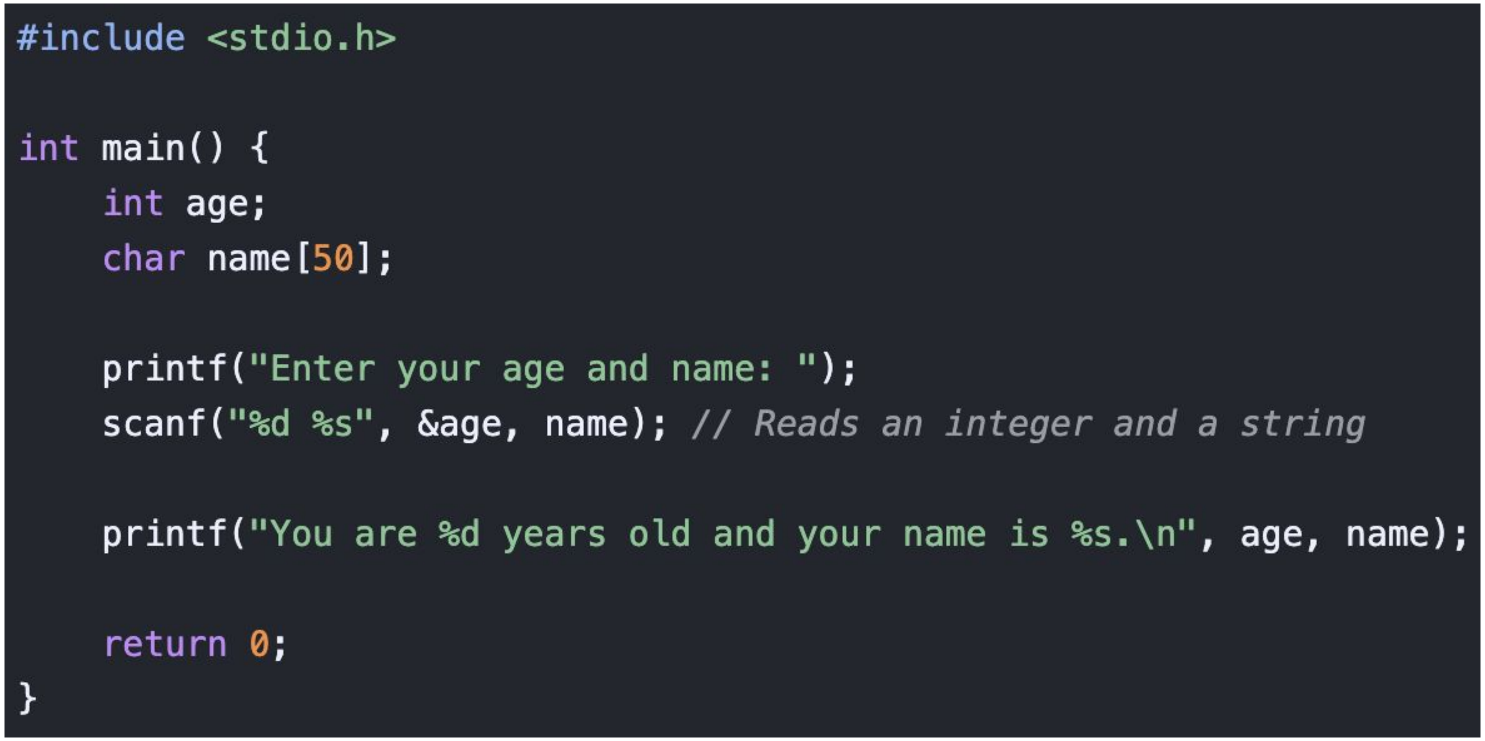
C Input - Scan Family

- Stream specifically formatted input
  - **scanf()**
    - reads from standard input stream
  - **fscanf()**
    - reads from input file
  - **sscanf()**
    - reads from character string

- Good for:
  - Input whose shape in predetermined
  - strips whitespace
- Common pitfalls:
  - expects <u>very specific</u> formatting
  - does not print error messages or clear remaining input buffer

C Input - Scan Family

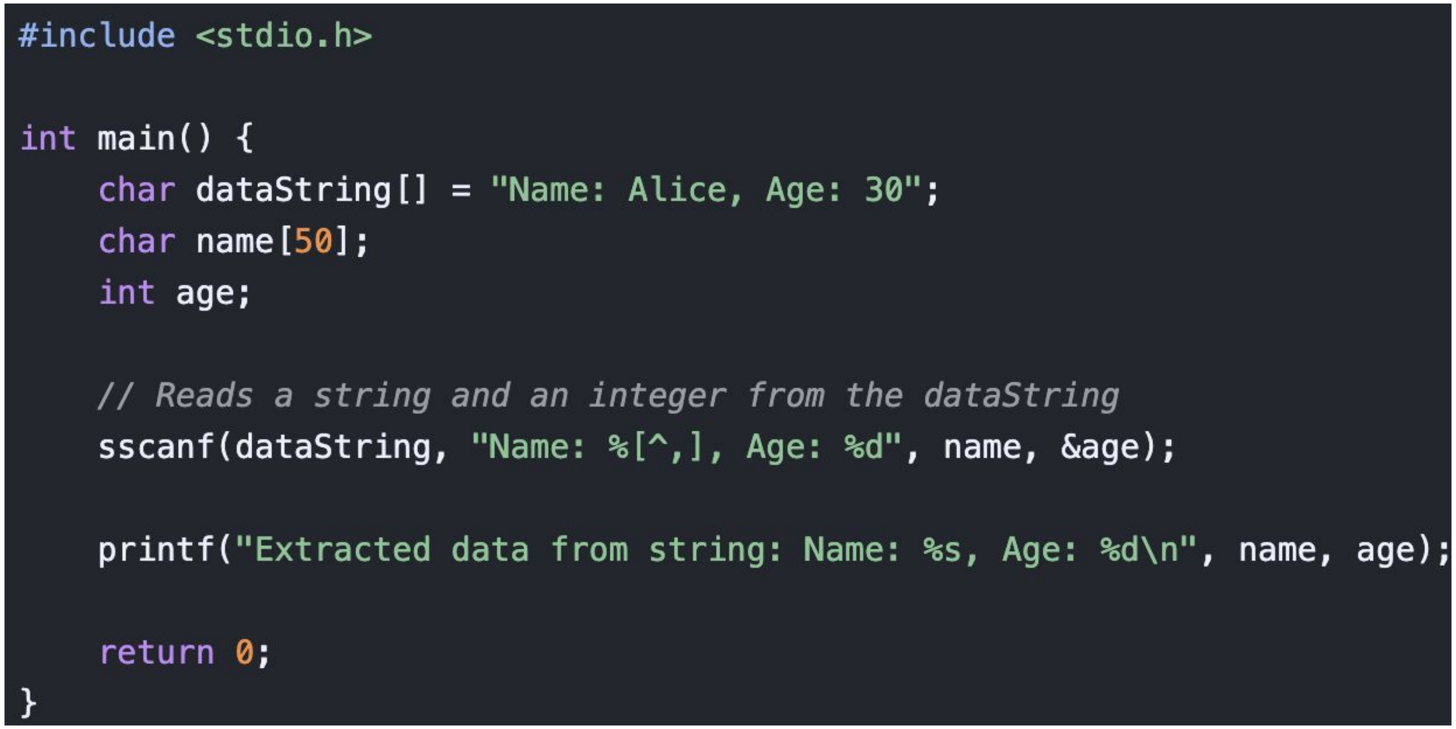
- **scanf()**
  - reads from standard input stream



```c
#include <stdio.h>

int main() {
    int age;
    char name[50];

    printf("Enter your age and name: ");
    scanf("%d %s", &age, name); // Reads an integer and a string

    printf("You are %d years old and your name is %s.\n", age, name);

    return 0;
}
```

C Input - Scan Family

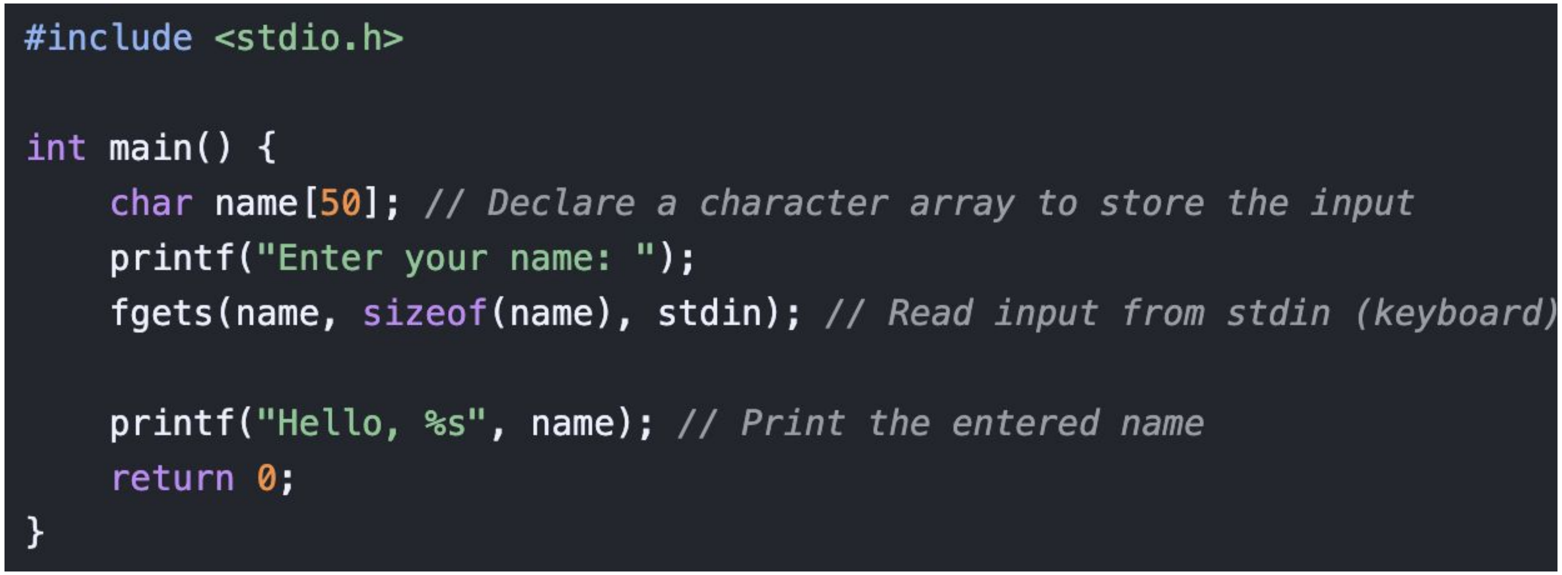
- **sscanf()**
  - reads from character string



```c
#include <stdio.h>

int main() {
    char dataString[] = "Name: Alice, Age: 30";
    char name[50];
    int age;

    // Reads a string and an integer from the dataString
    sscanf(dataString, "Name: %[^,], Age: %d", name, &age);

    printf("Extracted data from string: Name: %s, Age: %d\n", name, age);

    return 0;
}
```

C Input - Get Family

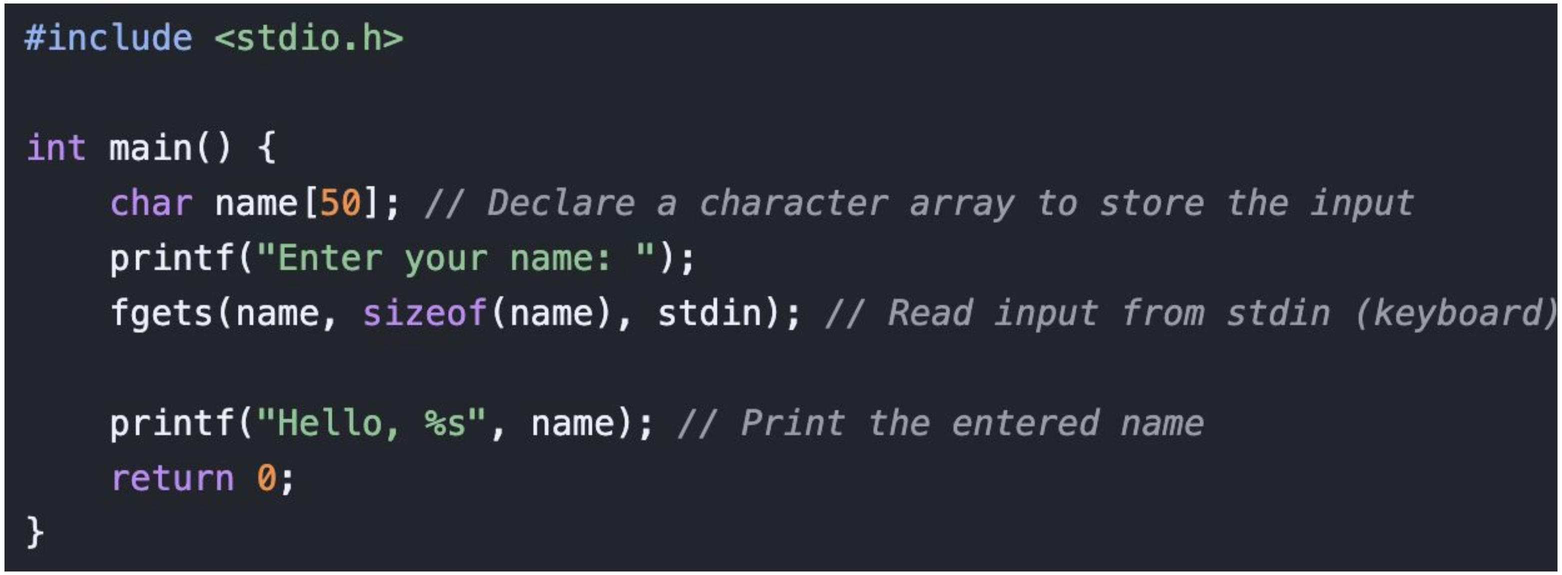
- fgets() - reads single line from user input
  - does not parse input
  - consumes entire line (up to size-1 or endline)
  - storage bound by buffer size
  - predefined size
  - automatic memory management



```c
#include <stdio.h>

int main() {
    char name[50]; // Declare a character array to store the input
    printf("Enter your name: ");
    fgets(name, sizeof(name), stdin); // Read input from stdin (keyboard)

    printf("Hello, %s", name); // Print the entered name
    return 0;
}
```

C Input - fgets()

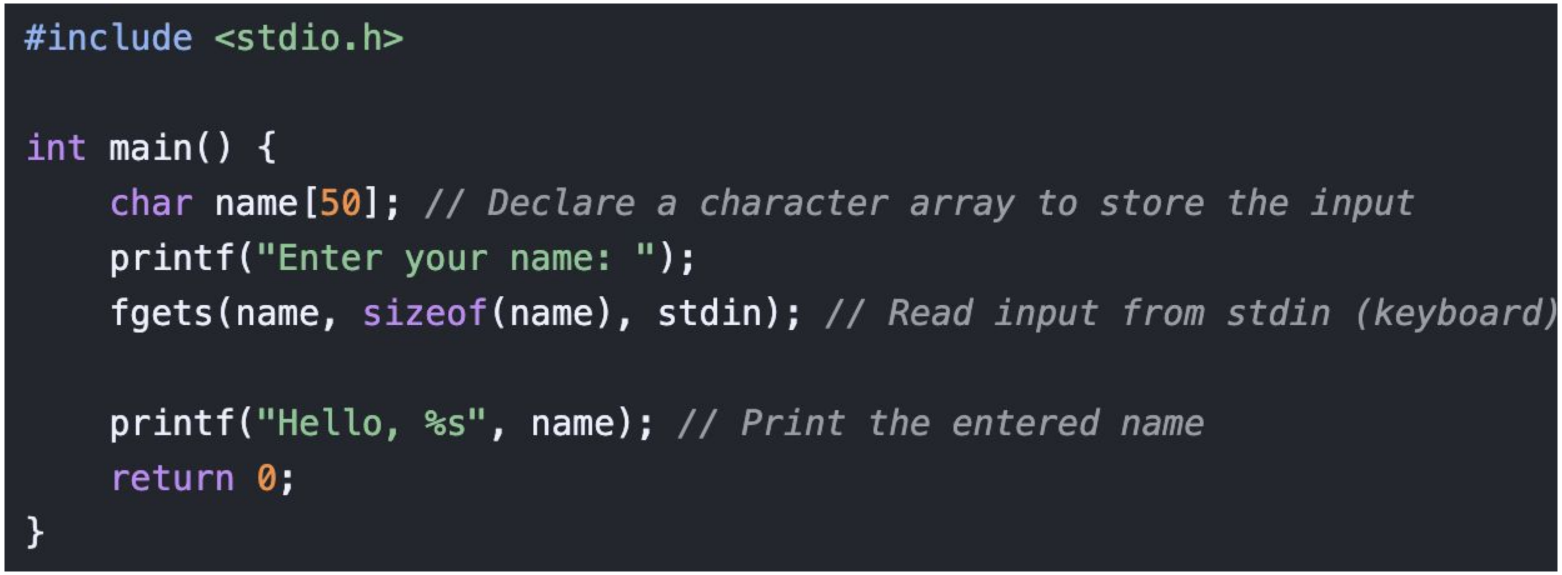
- fgets() common pitfalls:
  - no parsing
    - will often need to combine with another method
  - text beyond buffer size is left for next fgets()



```c
#include <stdio.h>

int main() {
    char name[50]; // Declare a character array to store the input
    printf("Enter your name: ");
    fgets(name, sizeof(name), stdin); // Read input from stdin (keyboard)

    printf("Hello, %s", name); // Print the entered name
    return 0;
}
```

C Input - fgets()

- fgets() common pitfalls:
  - no parsing
    - will often need to combine with another method
  - text beyond buffer size is left for next fgets()



```c
#include <stdio.h>

int main() {
    char name[50]; // Declare a character array to store the input
    printf("Enter your name: ");
    fgets(name, sizeof(name), stdin); // Read input from stdin (keyboard)

    printf("Hello, %s", name); // Print the entered name
    return 0;
}
```

C Input - getline()

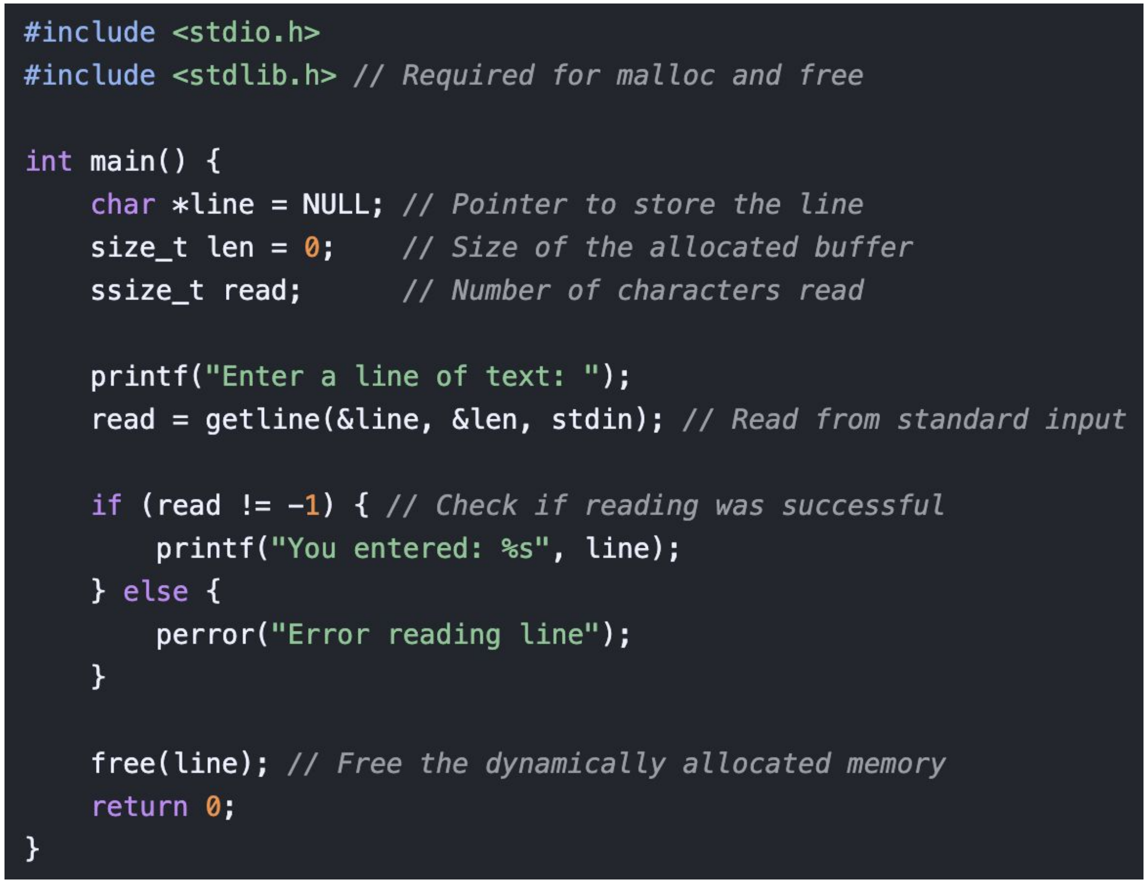
- getline() - retrieve full line of text
  - returns # of bytes read on success

- Getline is good for:
  - flexibility
  - working with either stdin or files
  - returns -1 to indicate EOF, time to exit

- Common pitfalls:
  - more variables
  - manual memory management

C Input - getline()

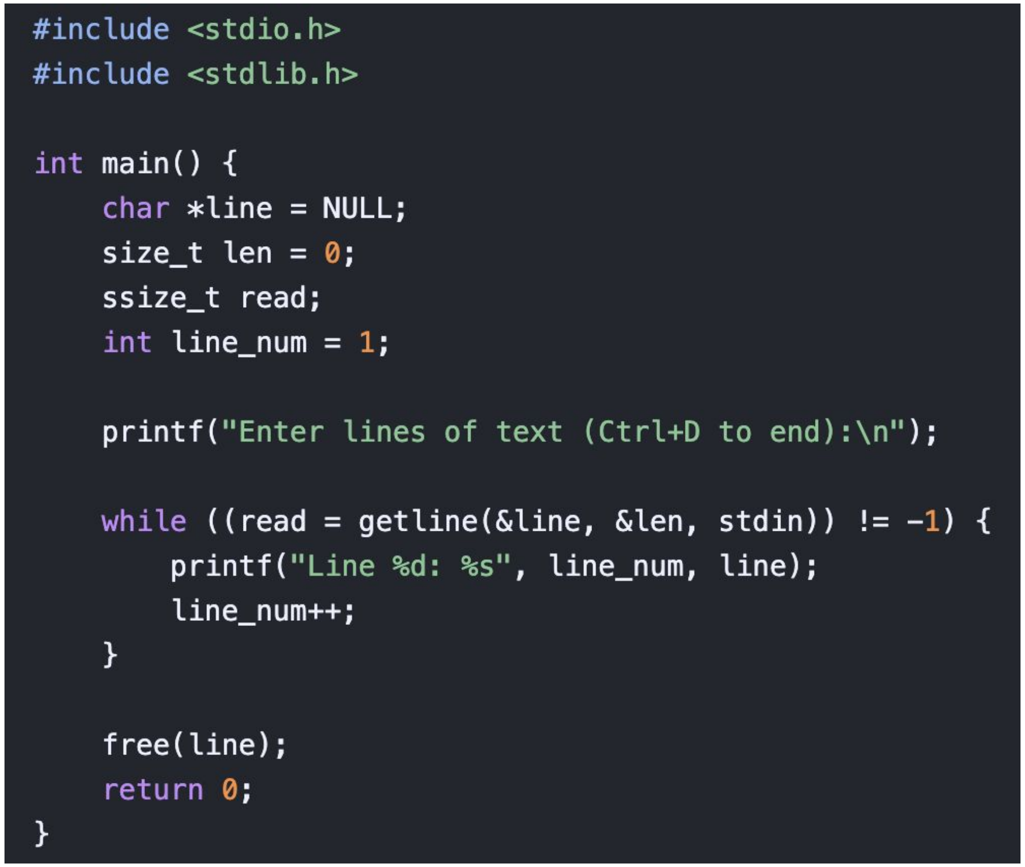
- getline() - retrieve full line of text



```c
#include <stdio.h>
#include <stdlib.h> // Required for malloc and free

int main() {
    char *line = NULL; // Pointer to store the line
    size_t len = 0;    // Size of the allocated buffer
    ssize_t read;      // Number of characters read

    printf("Enter a line of text: ");
    read = getline(&line, &len, stdin); // Read from standard input

    if (read != -1) { // Check if reading was successful
        printf("You entered: %s", line);
    } else {
        perror("Error reading line");
    }

    free(line); // Free the dynamically allocated memory
    return 0;
}
```

C Input - getline()

- getline() - retrieve full line of text



```c
#include <stdio.h>
#include <stdlib.h>

int main() {
    char *line = NULL;
    size_t len = 0;
    ssize_t read;
    int line_num = 1;

    printf("Enter lines of text (Ctrl+D to end):\n");

    while ((read = getline(&line, &len, stdin)) != -1) {
        printf("Line %d: %s", line_num, line);
        line_num++;
    }

    free(line);
    return 0;
}
```

C Input - getchar()

- getchar() - retrieve single character from stdin
    - usually coming from keyboard
    - on success returns either ASCII code for char or EOF
    - consumes a single character, leaving the rest in stdin for subsequent calls

- Good for:
    - simple and efficient
    - reads every character

- Common pitfalls:
    - Doesn’t work as easily with large input
    - Doesn’t play as well with parsers as other methods
    - Return type is int not char

C Input - getchar()

- getchar() - retrieve single character from stdin
  - usually coming from keyboard
  - on success returns either ASCII code for char or EOF
  - consumes a single character, leaving the rest in stdin for subsequent calls
- Good for:
  - simple and efficient
  - reads every character
- Common pitfalls:
  - Doesn’t work as easily with large input
  - Doesn’t play as well with parsers as other methods
  - Return type is int not char

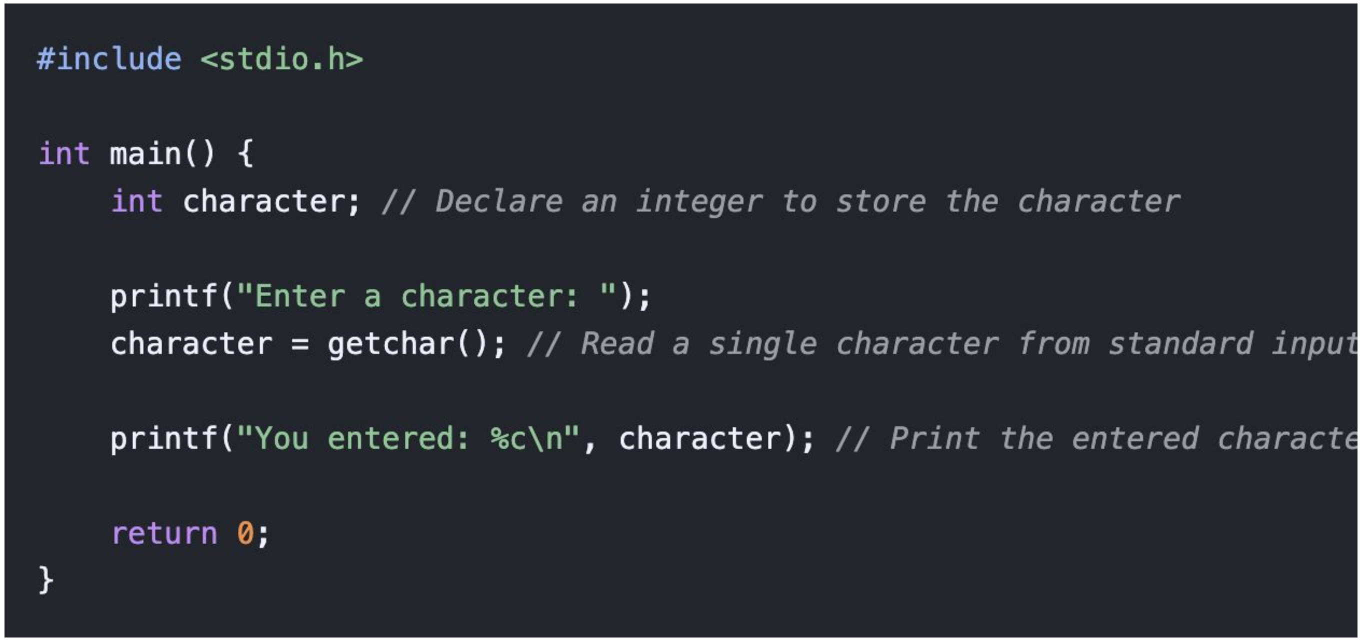
C Input - getchar()

- getchar() - retrieve single character from stdin
  - usually coming from keyboard
  - on success returns either ASCII code for char or EOF
  - consumes a single character, leaving the rest in stdin for subsequent calls

- Good for:
  - simple and efficient
  - reads every character

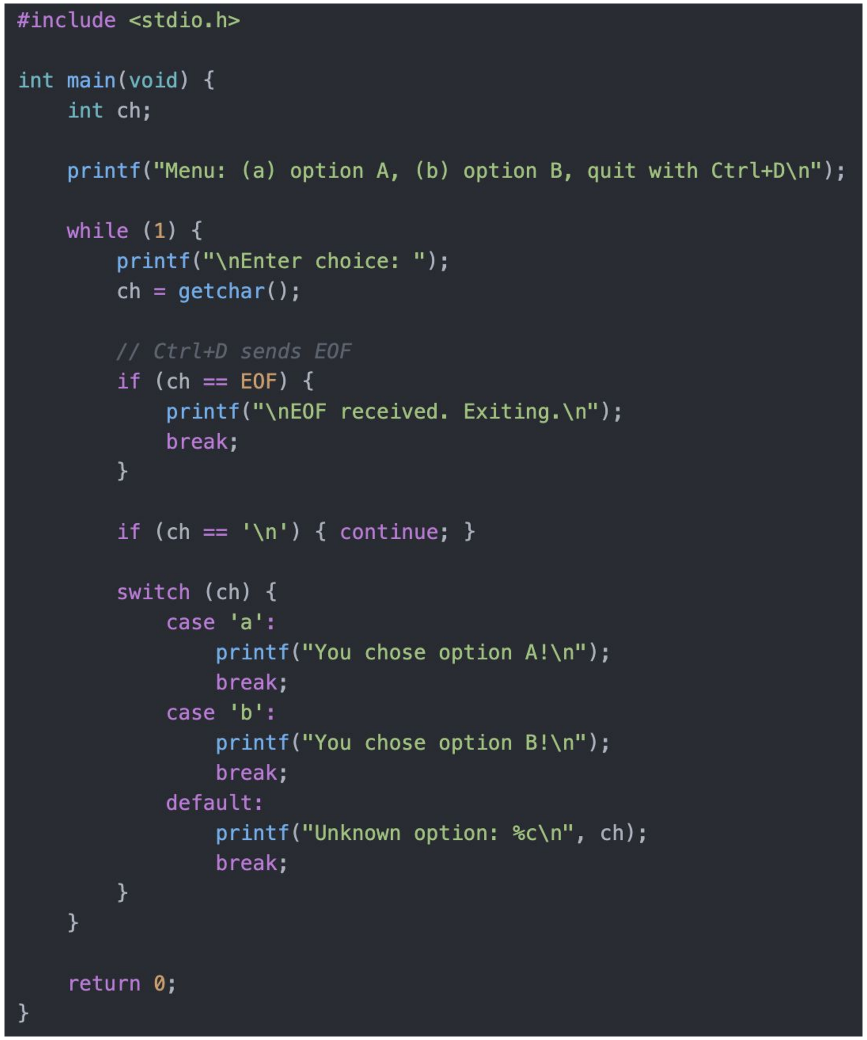
- Common pitfalls:
  - Doesn’t work as easily with large input
  - Doesn’t play as well with parsers as other methods
  - Return type is int not char

C Input - getchar()



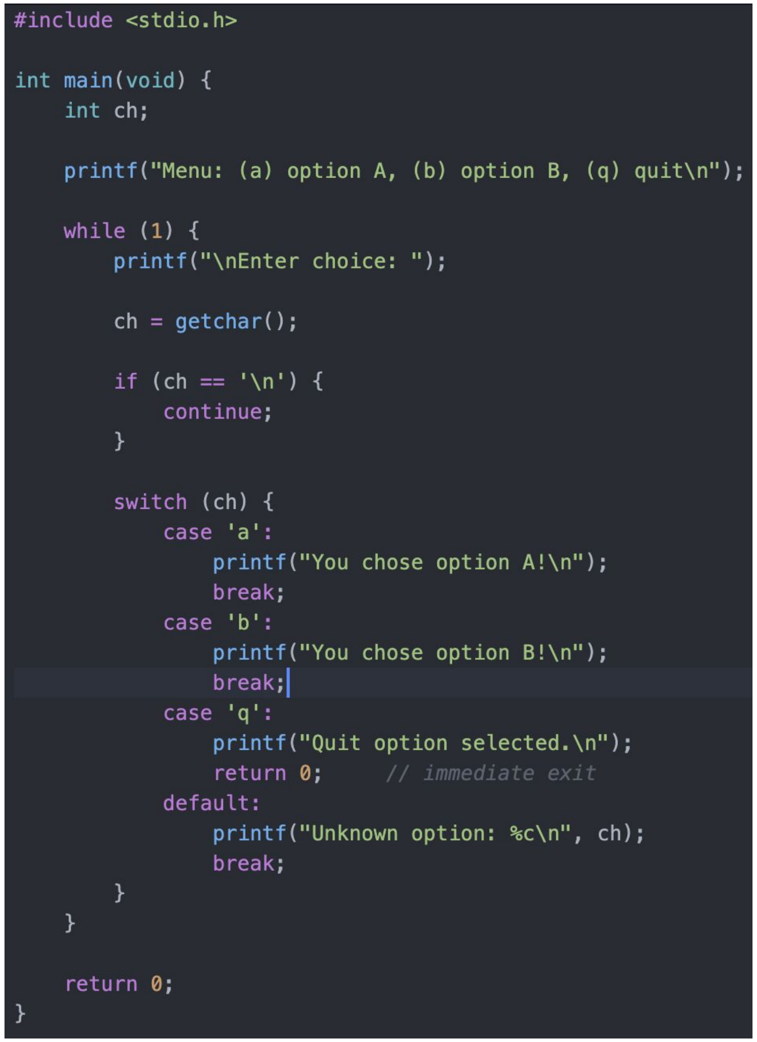
```c
#include <stdio.h>

int main() {
    int character; // Declare an integer to store the character

    printf("Enter a character: ");
    character = getchar(); // Read a single character from standard input

    printf("You entered: %c\n", character); // Print the entered characte

    return 0;
}
```

C Input - getchar()



```c
#include <stdio.h>

int main(void) {
    int ch;

    printf("Menu: (a) option A, (b) option B, quit with Ctrl+D\n");

    while (1) {
        printf("\nEnter choice: ");
        ch = getchar();

        // Ctrl+D sends EOF
        if (ch == EOF) {
            printf("\nEOF received. Exiting.\n");
            break;
        }

        if (ch == '\n') { continue; }

        switch (ch) {
            case 'a':
                printf("You chose option A!\n");
                break;
            case 'b':
                printf("You chose option B!\n");
                break;
            default:
                printf("Unknown option: %c\n", ch);
                break;
        }
    }

    return 0;
}
```

C Input - getchar()



```c
#include <stdio.h>

int main(void) {
    int ch;

    printf("Menu: (a) option A, (b) option B, (q) quit\n");

    while (1) {
        printf("\nEnter choice: ");

        ch = getchar();

        if (ch == '\n') {
            continue;
        }

        switch (ch) {
            case 'a':
                printf("You chose option A!\n");
                break;
            case 'b':
                printf("You chose option B!\n");
                break;
            case 'q':
                printf("Quit option selected.\n");
                return 0;     // immediate exit
            default:
                printf("Unknown option: %c\n", ch);
                break;
        }
    }

    return 0;
}
```

C File Access

- fopen() - native to C, returns FILE*, buffered I/O
  - c wrappers
  - pairs well with fgets(), fread(), fprintf()
  - flags: "r" "w" "a" "r+" "w+" "a+"

- open() - linux, returns int file descriptor, offers greater control
  - system calls
  - pair well with read(), write(), lseek()
  - modes:O_RDONLY, O_WRONLY, O_RDWR, O_CREAT, O_EXCL, O_TRUNC, O_APPEND

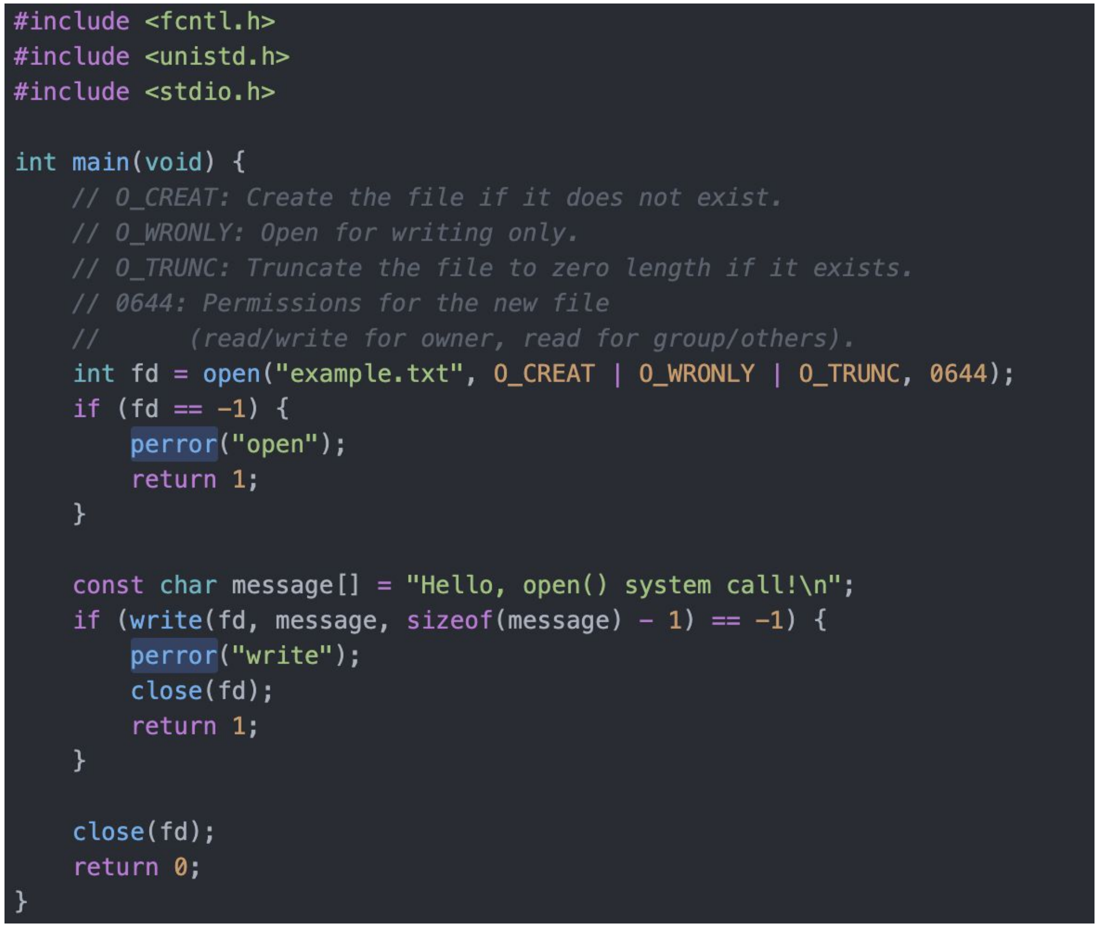
C File Access

- fopen() - native to C, returns FILE*, buffered I/O
    - c wrappers
    - pairs well with fgets(), fread(), fprintf()
    - flags: "r" "w" "a" "r+" "w+" "a+"

- open() - linux, returns int file descriptor, offers greater control
    - system calls
    - pair well with read(), write(), lseek()
    - modes:O_RDONLY, O_WRONLY, O_RDWR, O_CREAT, O_EXCL, O_TRUNC, O_APPEND

C File Access

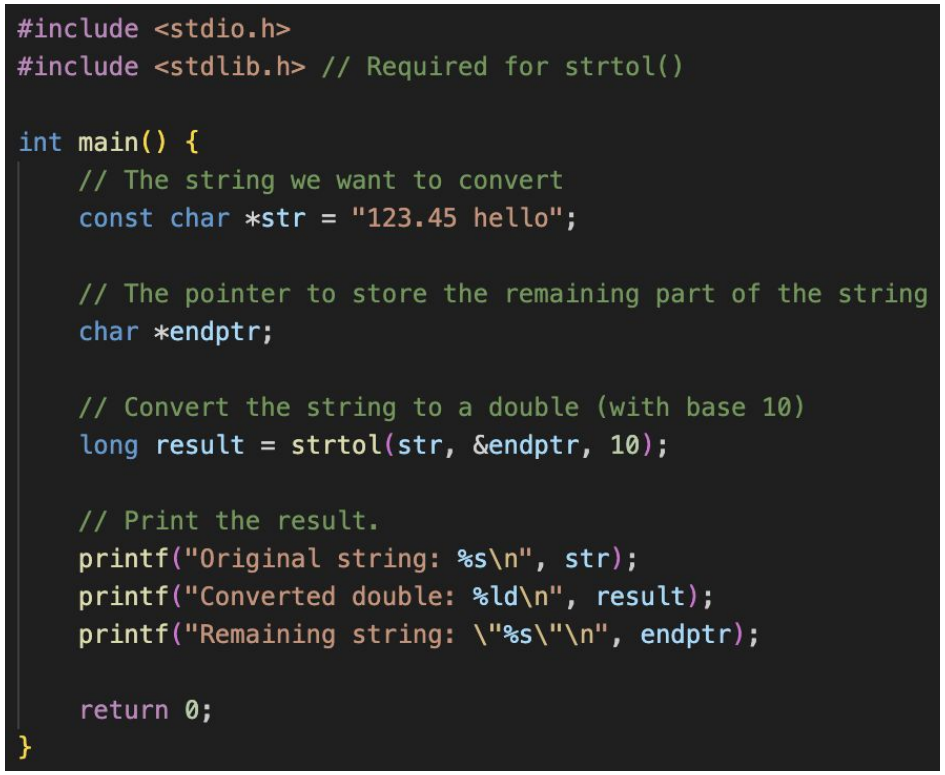
- open() example



```c
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>

int main(void) {
    // O_CREAT: Create the file if it does not exist.
    // O_WRONLY: Open for writing only.
    // O_TRUNC: Truncate the file to zero length if it exists.
    // 0644: Permissions for the new file
    //       (read/write for owner, read for group/others).
    int fd = open("example.txt", O_CREAT | O_WRONLY | O_TRUNC, 0644);
    if (fd == -1) {
        perror("open");
        return 1;
    }

    const char message[] = "Hello, open() system call!\n";
    if (write(fd, message, sizeof(message) - 1) == -1) {
        perror("write");
        close(fd);
        return 1;
    }

    close(fd);
    return 0;
}
```

C Numeric Parsing

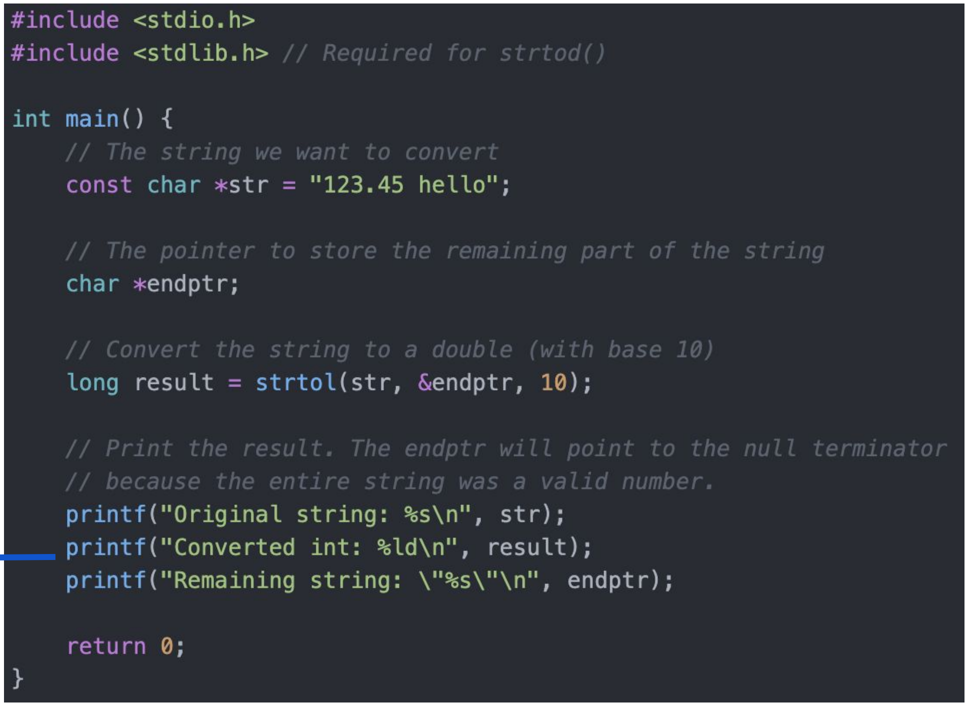
- We often need to convert strings into numeric datatype
  - strtol() - string to (long) int
  - strtoll() - string to long long
  - strtod() - string to double



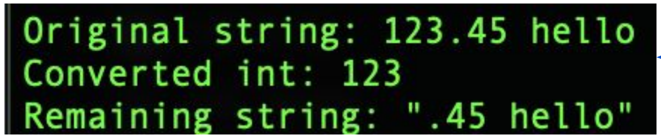
```c
#include <stdio.h>
#include <stdlib.h> // Required for strtol()

int main() {
    // The string we want to convert
    const char *str = "123.45 hello";

    // The pointer to store the remaining part of the string
    char *endptr;

    // Convert the string to a double (with base 10)
    long result = strtol(str, &endptr, 10);

    // Print the result.
    printf("Original string: %s\n", str);
    printf("Converted double: %ld\n", result);
    printf("Remaining string: \"%s\"\n", endptr);

    return 0;
}
```

C Numeric Parsing

- We often need to convert strings into numeric datatype
  - strtol() - string to (long) int
  - strtoll() - string to long long
  - strtod() - string to double



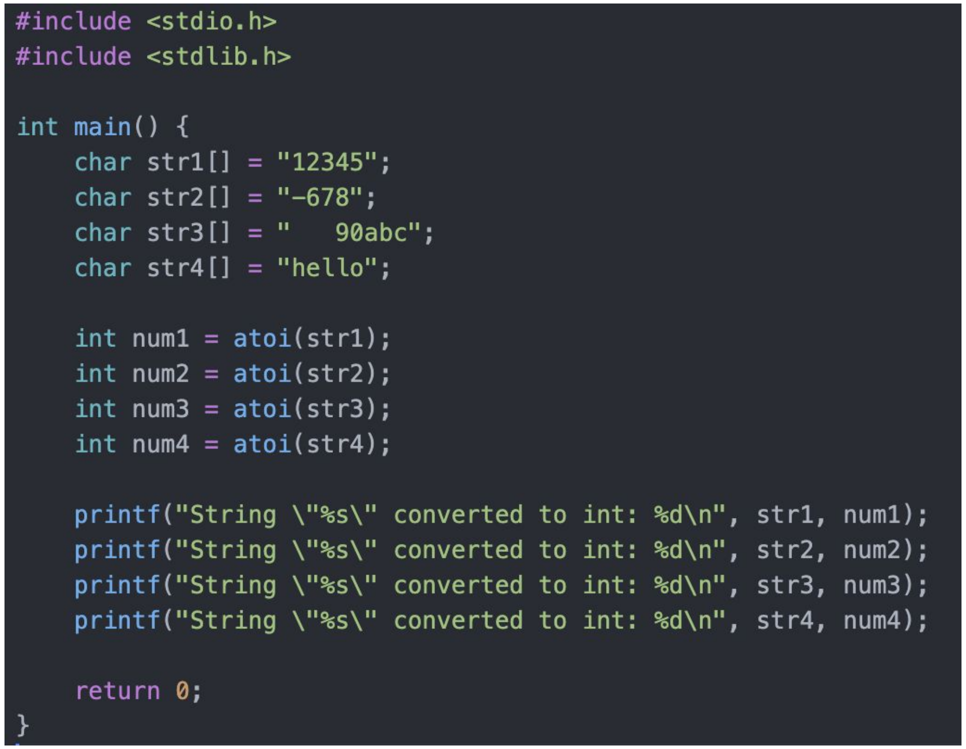
```c
#include <stdio.h>
#include <stdlib.h> // Required for strtod()

int main() {
    // The string we want to convert
    const char *str = "123.45 hello";

    // The pointer to store the remaining part of the string
    char *endptr;

    // Convert the string to a double (with base 10)
    long result = strtol(str, &endptr, 10);

    // Print the result. The endptr will point to the null terminator
    // because the entire string was a valid number.
    printf("Original string: %s\n", str);
    printf("Converted int: %ld\n", result);
    printf("Remaining string: \"%s\"\n", endptr);

    return 0;
}
```



```
Original string: 123.45 hello
Converted int: 123
Remaining string: ".45 hello"
```

C Numeric Parsing

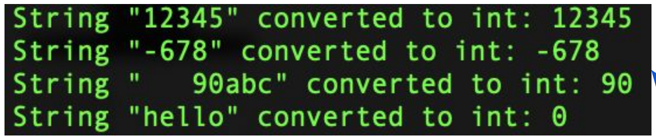
- We often need to convert strings into numeric datatype
  - atoi() - ascii to int
    - easier to use, but less configurable and doesn't return error messages



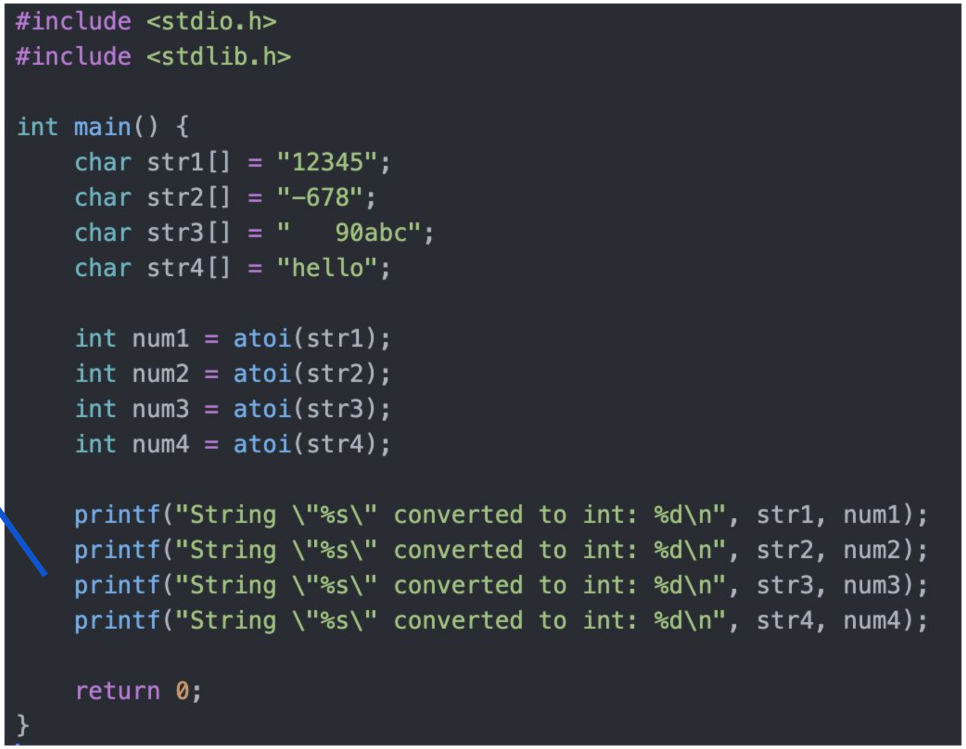
```c
#include <stdio.h>
#include <stdlib.h>

int main() {
    char str1[] = "12345";
    char str2[] = "-678";
    char str3[] = "   90abc";
    char str4[] = "hello";

    int num1 = atoi(str1);
    int num2 = atoi(str2);
    int num3 = atoi(str3);
    int num4 = atoi(str4);

    printf("String \"%s\" converted to int: %d\n", str1, num1);
    printf("String \"%s\" converted to int: %d\n", str2, num2);
    printf("String \"%s\" converted to int: %d\n", str3, num3);
    printf("String \"%s\" converted to int: %d\n", str4, num4);

    return 0;
}
```

C Numeric Parsing

- We often need to convert strings into numeric datatype
  - atoi() - ascii to int
    - easier to use, but less configurable and doesn't return error messages



```
String "12345" converted to int: 12345
String "-678" converted to int: -678
String "   90abc" converted to int: 90
String "hello" converted to int: 0
```



```c
#include <stdio.h>
#include <stdlib.h>

int main() {
    char str1[] = "12345";
    char str2[] = "-678";
    char str3[] = "   90abc";
    char str4[] = "hello";

    int num1 = atoi(str1);
    int num2 = atoi(str2);
    int num3 = atoi(str3);
    int num4 = atoi(str4);

    printf("String \"%s\" converted to int: %d\n", str1, num1);
    printf("String \"%s\" converted to int: %d\n", str2, num2);
    printf("String \"%s\" converted to int: %d\n", str3, num3);
    printf("String \"%s\" converted to int: %d\n", str4, num4);

    return 0;
}
```

C Stdin / Stdout Demo!

- Download it here!